Preguntas Frecuentes (FAQ)

Instalación y Setup

P: ¿Cómo configuro mi máquina para desarrollar en PAE?

R: Sigue environment-setup.md. Necesitas:

- JDK 17
- Android Studio
- Android SDK 27-36
- Git

```
./gradlew clean build  # verificar setup
```

P: ¿Qué versión de Android Studio necesito?

R: Android Studio 2023.2 o superior. Si es vieja, actualiza:

- macOS: `brew install android-studio`
- Windows: descarga de developer.android.com

Compilación y Deployment

P: ¿Cómo compilo para release?

R: Ver build-and-deployment.md:

```
export KEYSTORE_PASS="tu_pass"
export KEY_PASS="tu_key_pass"

./gradlew :Machine:assembleRelease
# APK está en: build/outputs/apk/release/Machine-release.apk
```

P: ¿Por qué dice "Signing config not defined"?

R: Necesitas configurar signing en `build.gradle.kts`:

```
android {
    signingConfigs {
        release {
```

```
            storeFile = file("../keystore/machine-release.jks")
            storePassword = System.getenv("KEYSTORE_PASS")
            keyAlias = "machine-key"
            keyPassword = System.getenv("KEY_PASS")
        }
    }
    buildTypes {
        release {
            signingConfig = signingConfigs["release"]
        }
    }
}
```

P: ¿Cómo instalo en dispositivo?

R:

```
# Debug
adb install build/outputs/apk/debug/Machine-debug.apk

# Release (después de release build)
adb install build/outputs/apk/release/Machine-release.apk

# Reinstalar
adb install -r build/outputs/apk/release/Machine-release.apk
```

Desarrollo

P: ¿Cómo agrego un nuevo módulo?

R:

1. Crear carpeta: `mkdir MyModule`
2. Crear `MyModule/build.gradle.kts`:

```
plugins {
    id("com.android.library")
    kotlin("android")
}

android {
    namespace = "co.ada.mymodule"
    compileSdk = 36
    // ...
}
```

```
dependencies {
    implementation(project(":Contract"))
    implementation(project(":Core"))
}
```

1. Agregar a `settings.gradle.kts`:

```
include(":MyModule")
project(":MyModule").projectDir = file("MyModule")
```

1. Sync: `> Gradle → Sync Now`

P: ¿Cómo agrego código a Machine vs RutaPAE?

R:

- Machine (app Android): `Machine/src/main/…`
- RutaPAE (app Android): `RutaPAE/src/main/…`
- Lógica compartida: `MachineDomain/` o `RutaPAEDomain/`
- Datos: `MachineData/` o `RutaPAEData/`

P: ¿Dónde agrego un nuevo estado?

R: En `MachineDomain/src/…/state/`:

1. Crear `MyNewState.kt`
2. Implementar interfaz `State`
3. Agregar a lista en `StateManager.workflow`

Ver developer-guide.md.

P: ¿Cómo agrego un endpoint HTTP?

R: En `MachineDomain/src/…/endpoints/MachineEndpoints.kt`:

```
object MachineEndpoints {
    fun getMyData(): Request {
        return "${backendUrl}/api/mydata"
            .httpGet()
            .responseJson()
    }
}

// Usar
val data = MachineEndpoints.getMyData()
```

Testing

P: ¿Cómo escribo tests?

R: Ver testing-guide.md. Ejemplo unit test:

```
@RunWith(RobolectricTestRunner::class)
class StateManagerTest {
    @Test
    fun testStateTransition() {
        val manager = StateManager(mockRepository, mockHardware)
        manager.init(context)

        assertEquals("WaitingForWeight", manager.getCurrentState())
    }
}
```

Ejecutar:

```
./gradlew test
```

P: ¿Cómo hago UI tests?

R: Con Compose test:

```
@RunWith(AndroidJUnit4::class)
class LaboratoryScreenTest {
    @get:Rule
    val composeTestRule = createComposeRule()

    @Test
    fun testButtonVisible() {
        composeTestRule.setContent {
            LaboratoryScreen(viewModel())
        }
        composeTestRule.onNodeWithText("Capturar").assertIsDisplayed()
    }
}
```

P2P y Sincronización

P: ¿Cómo sincronizo entregas entre Machine y RutaPAE?

R: En RutaPAE, enqueue worker:

```
DeliverySyncScheduler.enqueue(context, machineId = 1L)
```

Esto crea una operación P2P que:

1. Descubre máquina
2. Conecta vía Wi-Fi Direct
3. Descarga entregas
4. Guarda localmente

Ver p2p-flow.md.

P: ¿Qué pasa si P2P falla?

R: Hay fallback a hotspot. Si también falla:

- Retry automático con backoff exponencial
- Notificación al usuario
- Peut être reintentado manualmente

P: ¿Cómo sé si máquina está disponible?

R:

```
val p2pManager = DomainManager.p2pManager
val peers = p2pManager.discoveredMachineCandidates()
// peers contiene máquinas disponibles
```

Hardware

P: ¿Cómo capturo foto en Machine?

R: En StateManager:

```
val bitmap = Camera2Service(context).takeSelfie()
if (bitmap != null) {
    stateData.facePhoto = bitmap
    next()
```

```
} else {
    retry("Camera error")
}
```

Ver hardware-integration.md.

P: ¿Cómo conecto a balanza Bluetooth?

R: Requiere permissions:

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
```

Luego:

```
val scale = BluetoothScaleManager(context, MAC_ADDRESS)
scale.addEventListener(listener)
val weight = scale.getWeight()
```

P: ¿Cómo obtengo MAC address de balanza?

R: En dispositivo Android:

```
adb shell bluetoothctl
> devices
# lista con direcciones MAC
```

Database y Persistencia

P: ¿Cómo hago query vectorial a beneficiarios?

R: Usar `findSimilar`:

```
val embedding = generateEmbedding(facePhoto)
val similar = BeneficiaryService.findSimilar(embedding, similarity = 0.8f)

// similar contiene top resultados por similitud
for (beneficiary in similar) {
    println("${beneficiary.name}: ${similarity}")
}
```

P: ¿Cómo agrego nuevo campo a Delivery?

R:

1. Editar `data class Delivery` en `MachineData/src/…/entities/`
2. El ORM migrará automáticamente (open-source VectorDB)

```
data class Delivery(
    val id: Long,
    val weight: Float,
    val nuevoCampo: String = ""  // new field
)
```

P: ¿Cómo elimino datos viejos?

R: Limpiar en background:

```
val oneMonthAgo = System.currentTimeMillis() - 30.days.inMilliseconds
repository.deliveries.where { it.createdAt < oneMonthAgo }
    .forEach { repository.deliveries.delete(it.id) }
```

Seguridad

P: ¿Cómo protejo credenciales sensitivas?

R: Ver security-and-privacy.md:

```
// Guardar tokens de forma segura
val encryptedPrefs = EncryptedSharedPreferences.create(context, ...)
encryptedPrefs.edit().putString("api_token", token).apply()

// Leer
val token = encryptedPrefs.getString("api_token", null)
```

P: ¿Cómo evito logging de datos sensibles?

R: Usar logger personalizado:

```
// ❌ Malo
Log.d("API", "Token: $token")

// ✅ Bueno
Log.d("API", "User authenticated")
```

```
// ✅ Mejor
SecureLogger.logToken(token)  // oculta la mayoría
```

P: ¿Mi app necesita certificate pinning?

R: Sí, en producción. Configurar:

```
val certificatePinner = CertificatePinner.Builder()
    .add("backend.api.com", "sha256/AAAAAAA...")
    .build()
```

Performance

P: ¿Cómo optimizo búsqueda de beneficiarios?

R: Este es el cuello de botella típico. Soluciones:

1. Índice vectorial HNSW (ya implementado in VectorDB)
2. Caché de resultados recientes
3. Limitar K nearest neighbors

P: ¿Por qué la app está lenta?

R: Causas comunes:

1. Operación BD en main thread - usar `withContext(Dispatchers.IO)`
2. Búsqueda sobre muchas fotos - implementar paginación
3. Memory leak - usar Android Profiler
4. Compilación debug - agregar -Xmx4g en gradle.properties

Ver troubleshooting.md#app-muy-lenta.

Debugging

P: ¿Cómo veo logs en Android Studio?

R: Abrir Logcat:

```
Android Studio → View → Tool Windows → Logcat
```

Filtrar:

```
Filter: "StateManager"  # solo líneas con StateManager
```

O vía terminal:

```
adb logcat | grep StateManager
```

P: ¿Cómo seteo breakpoint?

R: Click en número de línea (izquierda):

```
1. Marcar breakpoint (punto rojo)
2. Run → Debug 'Machine' (Shift+F9)
3. App pausa cuando llega a línea
4. F9 para continuar
5. F10 para step over
6. F11 para step into
```

P: App crashea pero sin error visible?

R: Ver logcat completo:

```
adb logcat | tail -100
```

O usar Android Profiler:

```
Android Studio → View → Tool Windows → Profiler
```

Errores comunes

P: "Gradle sync failed"

R:

```
./gradlew clean
rm -rf .gradle
./gradlew build
```

Ver troubleshooting.md#gradle-sync-failed.

P: "No such key: deliveries"

R: No synchronized gradle. Falta `sync`:

```
Android Studio: File → Sync Now

O terminal:
./gradlew clean build
```

P: "Camera permission denied"

R: Solicitar en runtime (API 23+):

```
if (ContextCompat.checkSelfPermission(context, Manifest.permission.CAMERA)
    != PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(activity,
arrayOf(Manifest.permission.CAMERA), 100)
}
```

P: "Bluetooth connection failed"

R:

1. Verificar device está pareado
2. Permisos solicitados
3. MAC address correcto
4. Dispositivo no está dormido

Ver troubleshooting.md#bluetooth-connection-refused.

Mejores prácticas

P: ¿Cuándo usar Flow vs StateFlow?

R:

- **Flow**: observable, emite datos secuencialmente, puede estar vacío
- **StateFlow**: siempre tiene estado actual, último valor disponible

```
// Flow: observable stream
val deliveriesFlow: Flow<List<Delivery>> = ...

// StateFlow: estado actual
```

```
val currentState = MutableStateFlow<State>(State.Idle)
currentState.value  // acceder estado actual
```

P: ¿Cuándo usar suspend vs async?

R:

- **suspend**: función que puede pausarse (estructura lineal)
- **async**: inicia tarea concurrent (necesita .await())

```
// suspend: simple
suspend fun getData(): String = repository.getString()

// async: paralelismo
val result1 = async { repository.getString() }
val result2 = async { repository.getInt() }
// esperar ambas
val r1 = result1.await()
val r2 = result2.await()
```

P: ¿Cómo organizo código nuevo?

R: Seguir estructura:

```
co.ada.MODULE/
├── screens/       # UI Composables (si app)
├── viewmodels/    # ViewModel (si app)
├── domain/
│   ├── managers/  # orquestadores
│   └── services/  # lógica
├── data/
│   ├── entities/  # modelos BD
│   └── repository/ # CRUD
└── di/            # inyección
```

Contribución

P: ¿Cómo contribuyo code?

R:

1. Fork / branch
2. Hacer cambios + tests
3. Push
4. Pull request con descripción

Ver best-practices.md#version-control.

P: ¿Qué estándares de código?

R: Ver best-practices.md:

- Usar nombres descriptivos
- One responsibility per class/function
- Test coverage mínimo 70%
- Sin hardcoded secrets
- Lint checks pasan

Contacto y Soporte

P: ¿Dónde reporto bugs?

R: Issues en GitHub con:

- Descripción clara
- Steps para reproducir
- Log/stack trace
- Ambiente (dispositivo, OS, versión app)

P: ¿Dónde veo roadmap?

R: Proyectos en GitHub. Planificación trimestral.

P: ¿Cómo pido una feature?

R: Abrir discussion en GitHub con:

- Caso de uso
- Por qué es importante
- Impacto estimado

Recursos

- Documentación completa
- Architecture Decision Records
- Best practices

- Troubleshooting
- Glosario
- API Reference